(19)日本国特許庁（ＪＰ）　(12) **公開特許公報（Ａ）**　(11)特許出願公開番号
特開2002－163246
（P2002－163246A）
(43)公開日　平成14年６月７日(2002.6.7)

(51)Int.Cl.⁷	識別記号	ＦＩ	テーマコード（参考）
Ｇ０６Ｆ　17/12		Ｇ０６Ｆ　17/12	５Ｂ０４５
15/16	６１０	15/16	６１０Ｚ　５Ｂ０５６

審査請求　未請求　請求項の数７　ＯＬ　（全 12 頁）

(21)出願番号　特願2000－358232(P2000－358232)

(22)出願日　平成12年11月24日(2000.11.24)

(71)出願人　000005223
富士通株式会社
神奈川県川崎市中原区上小田中４丁目１番１号

(72)発明者　中西　誠
神奈川県川崎市中原区上小田中４丁目１番１号　富士通株式会社内

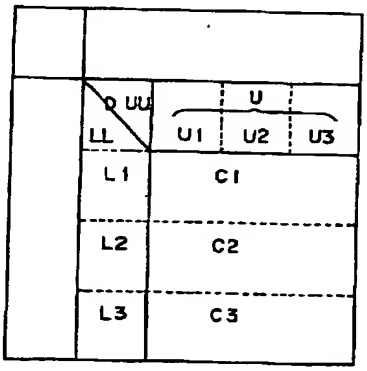
(74)代理人　100074099
弁理士　大菅　義之　（外１名）

Ｆターム(参考)　5B045 AA07 DD02 DD12 GG11
5B056 AA04 BB02

(54)【発明の名称】　共有メモリ型スカラ並列計算機における並列行列処理方法、及び記録媒体

(57)【要約】
【課題】共有メモリ型スカラ計算機に適した並列行列処理方法を提供する。
【解決手段】ＬＵ分解すべき行列を対角部分のブロックＤとＤの下側にある列方向のブロック、例えば、Ｌ１～Ｌ３に分ける。そして、３つのプロセッサのそれぞれに、Ｄ＋Ｌ１、Ｄ＋Ｌ２、Ｄ＋Ｌ３を割り振り、並列して演算を行わせる。そして、ブロックＵをＬＵ分解の方法によって更新し、更に、Ｌ１～Ｌ３とＵとを用いてＣ１～Ｃ３を更新する。この処理を順次小さくなっていく内側もブロックに施すことにより、最後に、Ｄに対応する部分だけが残る。このＤを１つのプロセッサでＬＵ分解することにより行列全体のＬＵ分解を終える。

本発明の実施形態に従ったLU分解の並列処理の概念を説明する図（その1）



【特許請求の範囲】
【請求項１】複数のプロセッサモジュールを持つ共有メモリ型スカラ並列計算機の行列演算において、
行列を小行列ブロックに分けるブロック化ステップと、
該小行列ブロックの内、対角ブロックと対角ブロックでない小行列ブロックとを該複数のプロセッサモジュールのローカルメモリ領域に格納する格納ステップと、
該複数のプロセッサモジュールが並列にそれぞれ有するブロックを演算することにより、複数のプロセッサモジュールで、対角ブロックを冗長に演算する演算ステップと、
該演算ステップで得られた小行列ブロックの演算結果を使って、該行列を更新する更新ステップと、を備えることを特徴とする並列行列処理方法を情報装置に実現させるプログラムを格納した、情報装置読み取り可能な記録媒体。
【請求項２】該行列演算は、行列のＬＵ分解であることを特徴とする請求項１に記載の記録媒体。
【請求項３】前記複数のプロセッサモジュールが有する小行列ブロックのデータからそれぞれがピボットの候補を抽出する抽出ステップと、
該ピボットの候補から該複数のプロセッサモジュールに共通のメモリ領域において、データ値が最大値を示すピボット候補を最終的なピボットと決定するピボット決定ステップと、を更に備え、該決定されたピボットを用いてＬＵ分解を行うことを特徴とする請求項２に記載の記録媒体。
【請求項４】前記ＬＵ分解は、前記行列の外側から再帰的なアルゴリズムによって順次行列の更新を行い、前記行列の内、最後に更新し残った部分を、１つのプロセッサモジュールでＬＵ分解することにより、該行列全体についてＬＵ分解を完了することを特徴とする請求項２に記載の記録媒体。
【請求項５】該行列演算は、行列のコレスキー分解あるいは変形コレスキー分解であることを特徴とする請求項１に記載の記録媒体。
【請求項６】前記コレスキー分解あるいは変形コレスキー分解は、前記行列の外側から再帰的なアルゴリズムによって順次行列の更新を行い、前記行列の内、最後に更新し残った部分を、１つのプロセッサモジュールでＬＵ分解することにより、該行列全体についてＬＵ分解を完了することを特徴とする請求項２に記載の記録媒体。
【請求項７】前記更新ステップにおいて、更新すべき小行列ブロックの三角行列部分を前記複数のプロセッサモジュールの数の２倍の数のブロックに分割し、該分割された三角行列部分のブロックを２つずつ組み合わせて、各プロセッサモジュールのローカルメモリ域に格納し、演算をプロセッサモジュールに行わせることを特徴とする請求項５に記載の記録媒体。
【発明の詳細な説明】
【０００１】
【発明の属する技術分野】本発明は、共有メモリ型スカラ並列計算機における並列行列処理方法に関する。
【０００２】
【従来の技術】連立１次方程式を計算機によって解く場合には、連立１次方程式を行列表示し、この行列について処理を施すことによって、解の求めやすい形に変形し、このような形にしてから方程式の解を求める方法が採られている。
【０００３】すなわち、連立１次方程式は、係数を表す行列と、変数を表す列ベクトルとの積が所定の列ベクトルに等しくなるというように記述することが出来る。ここで、ＬＵ分解（LU Factorization）という方法によれば、係数を表す行列を上三角行列（upper-triangular matrix ）と下三角行列（lower-triangular matrix ）に分解することによって、連立１次方程式の解を求める。したがって、この場合においては、係数行列をＬＵ分解することが連立１次方程式の解を得るために重要な処理となる。また、ＬＵ分解の特別な場合として（変形）コレスキー分解（Cholesky Factorization）という行列の分解方法がある。
ａ）実行列の連立１次方程式の解法
実行列の連立１次方程式の解法に関して、ベクトル並列計算機での連立１次方程式は、ブロック化した外積型のＬＵ分解をベースに並列化を行っている。列ベクトルを何本か束ねたブロックをＬＵ分解（１）した後、対応した行ベクトルを束ねたブロックを更新して（２）から、正方小行列を更新（３）する処理を繰り返す。
【０００４】従来、（１）の処理は、一つのプロセッサで逐次的に行っていた。並列効率を高めるためにブロック幅は１２（列あるいは行：係数行列の行幅あるいは列幅を示す）程度の比較的小さい値としていた。この結果（２）及び（３）の部分の更新も幅１２程度の行列演算となった。
【０００５】最もコストの大きい（３）の計算で、幅が１２程度と小さくても、効率の良い方法があった。共有メモリ型スカラ並列計算機（ＳＭＰ）では、幅が小さいと性能を引き出せない。これは以下の理由による。
【０００６】すなわち、（３）の計算は、行列積である。幅が小さいと更新する行列の要素（メモリに格納されている）をロードして、更新結果をストアするというメモリにアクセスするためのコストが行列の更新を行う演算に比べて大きくなり性能を引き出せない。
【０００７】このため、ブロック幅を大きくする必要があるが、ブロック幅を大きくすると、このブロックのＬＵ分解のコストが大きくなり並列化効率が落ちる。
ｂ）正値対称行列の連立１次方程式の解法
正値対称行列の連立１次方程式の解法に関して、下三角行列部分のみを利用してコレスキー分解を行うときに、分散メモリ型並列計算機では小行列ブロックをｃｙｃｌ

ｉｃに各プロセッサに分散して負担させ、各プロセッサの負荷を均等にして解いていた。実行列の連立１次方程式と同じようにブロック化するときブロック幅を比較的小さくでき、並列化効率を高めることが可能であった。ＳＭＰでは、上記（３）の更新で現れる行列積でブロック幅が大きい方が性能が良いため、ブロック幅を大きくする必要がある。

【０００８】

【発明が解決しようとする課題】すなわち、共有メモリ型スカラ並列計算機では、ＬＵ分解あるいはコレスキー分解における行列積による行列の更新に必要とされるコストよりも、共有メモリにアクセスするためのコストが大きくなってしまい、従来ベクトル並列計算機で行っていた方法をそのまま共有メモリ型スカラ計算機に適用しても十分な性能を引き出せない。

【０００９】本発明の課題は、共有メモリ型スカラ計算機に適した並列行列処理方法を提供することである。

【００１０】

【課題を解決するための手段】並列行列処理方法は、複数のプロセッサモジュールを持つ共有メモリ型スカラ並列計算機の行列演算において、行列を小行列ブロックに分けるブロック化ステップと、該小行列ブロックの内、対角ブロックと対角ブロックでない小行列ブロックとを該複数のプロセッサモジュールのローカルメモリ領域に格納する格納ステップと、該複数のプロセッサモジュールが並列に、それぞれ有するブロックを演算することにより、複数のプロセッサモジュールで、対角ブロックを冗長に演算する演算ステップと、該演算ステップで得られた小行列ブロックの演算結果を使って、該行列を更新する更新ステップとを備えることを特徴とする。

【００１１】本発明によれば、複数のプロセッサモジュールがそれぞれ有するローカルメモリ領域を有効に利用して、効率的な行列演算を行うことが出来る。

【００１２】

【発明の実施の形態】図１は、共有メモリ型スカラ並列計算機のハードウェア構成例を示す図である。

【００１３】共有メモリ型スカラ並列計算機は、複数のプロセッサ１０－１、１０－２、・・・１０－ｎが２次キャッシュメモリ１３－１、１３－２、・・・１３－ｎを介して相互結合網１２に接続される。各プロセッサ１０－１、１０－２、・・・１０－ｎは、その内部あるいは、２次キャッシュメモリ１３－１、１３－２、・・・１３－ｎよりプロセッサ側に１次キャッシュメモリが設けられる。また、各プロセッサ１０－１、１０－２、・・・１０－ｎに共有となっているメモリモジュール１１－１、１１－２、・・・１１－ｎは、相互結合網１２を介してプロセッサ１０－１、１０－２、・・・１０－ｎがアクセス可能となってる。プロセッサ１０－１、１０－２、・・・１０－ｎがデータ処理を行う場合には、まず、メモリモジュール１１－１、１１－２、・・・１１－ｎから１つのプロセッサが担当するデータを２次キャッシュメモリ１３－１、１３－２、・・・１３－ｎに格納し、更に、２次キャッシュメモリから処理単位となるデータを１次キャッシュメモリにコピーして処理を行う。

【００１４】処理が終わると、１次キャッシュメモリから２次キャッシュメモリに処理データが格納され、２次キャッシュメモリ内のデータが全て処理し終わると、メモリモジュール１１－１、１１－２、・・・１１－ｎの内、最初にデータを持ってきたメモリモジュールに対してデータの更新を行う。また、次のデータ処理を行う場合には、上述したように、メモリモジュールから各プロセッサが担当する分のデータを２次キャッシュメモリに格納し、１次キャッシュメモリに処理単位のデータを持ってきて、プロセッサが処理を行う。このような処理を繰り返して、並列にデータ処理を完了する。このとき、各プロセッサが処理した後のデータをメモリモジュールに書き込み、次の処理のために、再びメモリモジュールからデータを読み込む際、各プロセッサが自分のタイミングでデータの読み込みを行っていたのでは、データ更新された後のデータを読み込むべきところを、データ更新される前のデータを読み込んでしまう可能性が有る。したがって、このときには、全てのプロセッサがメモリモジュールにデータ更新し終わるまで、他のプロセッサがメモリモジュールからデータを読み込まないようにする必要がある。このように、プロセッサのメモリモジュールからのデータの読み込みを制限して、全体のプロセッサの処理の同期をとることをバリア同期（Barrier Synchronization）を取るという。

【００１５】図２及び図３は、本発明の実施形態に従ったＬＵ分解の並列処理の概念を説明する図である。図２は、処理すべき行列の模式図であり、図３は、処理単位となるデータの構成を説明する図である。

ａ）本実施形態に従った実行列の連立１次方程式の解法

本実施形態においては、プロセッサが処理を担当する小行列ブロックの幅を従来より大きくしてＬＵ分解する部分を並列化する。すなわち、図２において、従来では、Ｌ１～Ｌ３の部分は、ブロック幅を小さくして、１つのプロセッサ（ＰＥ、あるいは、スレッド）で処理していたが、本実施形態においては、このブロック幅を大きくすると共に、Ｌ１～Ｌ３をそれぞれ別のスレッドに割り当てて、並列に処理させる。なお、ここでは、スレッドの数を３つとしている。

【００１６】共有メモリ型スカラ並列計算機の各プロセッサは独立に１次及び２次キャッシュが備わっている。特に１次キャッシュ上のデータに載る範囲で、計算を行うことが高性能を引き出す上で重要である。

【００１７】データ量の大きな問題を多数ＰＥで解く場合、各ＰＥでデータを局所化して全体としてはブロックのＬＵ分解を並列に計算し、できるだけ大きなブロック

幅を確保することが必要となる。

【００１８】このために、図３に示されるように、各プロセッサでローカルに作業域を確保してＬ２キャッシュ（２次キャッシュ）に載る大きさで計算を行う。このとき、並列に更新するとき必要なブロック対角部分Ｄは、各ＰＥ（各スレッド）でコピーして各プロセッサ（各スレッド）で冗長に計算する。また、ＬＵ分解において、ピボットを取った後、行ベクトルの入れ替えを行う場合は、いずれかのプロセッサの２次キャッシュメモリ上に設けられた共用メモリ域を介して、各ＰＥで通信して、必要な情報の共用を行うようにする。

【００１９】ｂ：ブロック幅、ｋ：各プロセッサが分担する列ブロックの１次元目の大きさ（ブロック幅が小行列ブロックの列方向であるので、小行列ブロックの行の数、すなわち、図２におけるＬ１～Ｌ３のブロックの合計の行の数）としたとき、ｂ×（ｂ＋ｋ）×８～８Ｍｂｙｔｅを満たすものをブロック幅として採用する。

【００２０】そして、作業域（１次キャッシュメモリ）にコピーした部分に関して、キャッシュメモリ上のデータを利用したＬＵ分解を行う。また、処理すべき行列が占有するメモリ量が大きく（行列が大きく）、それに比べてプロセッサ数が少ないため、並列処理向けの列ブロックのブロック幅が小さくなるときは、ブロック幅を分割して必要なブロックを確保してから、行ブロックの更新と行列積の更新を並列に行う。

【００２１】更に、各プロセッサ（各スレッド）での作業域上でのＬＵ分解は、内積法など更新部分の内積ベクトルの長さが大きな方法を、例えば、アルゴリズムの再帰的な呼び出しで、更新部分の性能を引き出しながらＬＵ分解を行う方法を利用することで、キャッシュ上のデータを効率よく利用する。

【００２２】図４は、本実施形態のＬＵ分解の処理の流れを示す概略フローチャートである。まず、ステップＳ１において、スレッド数及び問題の大きさ（処理すべき行列の大きさ）からブロック幅を決定する。次に、ステップＳ２において、各スレッドが処理するブロックを決定し、各スレッドで処理するブロック（図２のＤ及びＬｉ）を作業域にコピーする。そして、ステップＳ３において、各スレッドでピボットを決定し、その中での最大値を示すピボットを共用域を使って決定し、最大値を示すピボットを用いて、行ベクトルを入れ替え、各スレッドで、上記ブロックＤとブロックＬｉとをＬＵ分解する。

【００２３】ステップＳ４においては、処理が終わりか否かを判断し、終わりの場合には、処理を終了するが、終わりでない場合には、ステップＳ５において、各スレッドで並列にブロックＬＬ（図２参照）を使って、ブロックＵｉ（図２参照）をＬＵ分解のアルゴリズムに従って更新する。そして、ステップＳ６において、各スレッドで並列に、ブロックＣｉ（図２参照）をブロックＬｉとブロックＵ（Ｕｉを組み合わせたもの：図２参照）の積で更新する。そして、ステップＳ２に戻り、次のブロック（Ｃｉからなるブロック）を、同様の方法でＬＵ分解する。そして、次第に小さくなる未処理ブロックが最後にブロックＤに対応する部分のみになり、１つのスレッドでＬＵ分解が完了すると、行列全体についてＬＵ分解が終了する。

【００２４】図５～図１０は、本実施形態のＬＵ分解の方法をより詳細に説明する図である。ここでは、２０４８×２０４８の行列を４スレッドでＬＵ分解する場合を例にとって説明する。

【００２５】まず、２０４８×２０４８の行列をブロック幅２５６でＬＵ分解するものとする。各ブロックの区分けは、図５に示すとおりである。そして、４つのＣＰＵ（スレッド）で処理を実行する場合、各スレッドで連続な領域（（２５６＋４４８）×２５６：８ＭＢ（Ｌ２キャッシュの大きさ）より小さい領域）を確保し、図６に示すように、各スレッドの各領域にＤ１＋Ｌ１、Ｄ１＋Ｌ２、Ｄ１＋Ｌ３、Ｄ１＋Ｌ４をコピーする。

【００２６】なお、ブロック幅は例えば、以下のようにして決める。問題の大きさ（行列の大きさ：行列の次数）をｎ、スレッドの数を＃ＴＨＲＥＡＤとして、

【００２７】

【数１】

$$\sqrt[3]{(n^2/\#\mathrm{THREAD})\times\frac{1}{100}}=nb$$

【００２８】とおいて、

ｎｂ≧５１２なら、ブロック幅＝５１２

ｎｂ≧２５６なら、ブロック幅＝２５６

ｎｂ≧１２８なら、ブロック幅＝１２８

それ以外、ブロック幅＝６４

というように、メニュー化しておき、この中から選ぶようにする。

【００２９】すなわち、ＬＵ分解のコストは、$2n^3/3$（ｎは行列の次数）で、n^3 に比例する。したがって、全体のコストを＃ＴＨＲＥＡＤで並列化し、その１％が最後に１スレッドで行うブロック幅程度となるように決める。

【００３０】ここで、理解を助けるため、並列化しない場合のＬＵ分解のアルゴリズムを図７に示す。図７においては、ＬＴ＝Ｄ１＋Ｌ１＋Ｌ２＋Ｌ３＋Ｌ４の部分をＬＵ分解するアルゴリズムを示している。ＬＴは、２０４８×２５６のブロックとなっている。

【００３１】まず、（１）の部分において、ピボットを決定する。iblks はＬＴの幅であり、今の場合、２５６である。また、lengは、ＬＴの長さであり、今の場合、２０４８である。ｊｊには、ピボットの存在する行番号が、ＴＭＰには、ピボットの絶対値が設定される。

【００３２】そして、（２）の部分において、現在処理しているＬＴ内の列番号ｉより、ピボットの存在する行

番号ｊｊが大きい場合に、ｉ番の行のデータをｊｊ番の行のデータと入れ替える。次に、（３）の部分で、列ｉのピボットを使って、ＬＵ分解の演算を行う。

【００３３】この（１）～（３）をｉが１～iblks にわたって繰り返し演算する。ここで、ＬＴの長さである、lengがもっと大きくなると、これらの処理はＬ２キャッシュメモリのデータを入れ替えてしまい、著しい性能低下を引き起こす。そこで、図５のように、データを分散し、Ｌ２キャッシュメモリにデータを保持したまま処理を行うようにする。各スレッドにおけるアルゴリズムは図８に示すとおりである。

【００３４】なお、図８においては、ＬＴｉは、ローカル域、pivot(4)、GPIVOT、ROW （iblks ）は、共用域に格納されるデータである。まず、（４）において、各スレッドでピボットを取る。そして、（５）で最大値を配列pivot(4)のスレッド番号の要素に格納する。（５）の後に、バリア同期を取って、（６）において、最大ピボットを持つスレッド番号をGPIVOTに格納する。そして、（６）の後で、再びバリア同期を取る。次に、（７）において、最大ピボットを持つスレッドが共用域ROWに最大ピボットの行ベクトルを格納し、バリア同期を取る。（８）においては、GPIVOTが０のときは、最大ピボットはＤ１の内にあるか、入れ替え不要であり、ローカル域に入れ替える。GPIVOTが＃ＴＨＲＥＡＤに等しいとき、すなわち、GPIVOTが０より大きい場合には、最大ピボットを持たないスレッドは、ROWの内容とｉ行目の行ベクトルを入れ替える。そして、（９）、（１０）において、ＬＵ分解のための演算を行う。そして、上記（４）～（１０）を処理するブロックの全ての列について行う。すなわち、１～iblks までのｉについて処理を繰り返す。

【００３５】ここで、ＬＴｉのＬＵ分解の最後でバリア同期を取る。そのあと、各スレッドのＤ１の部分はＬＬとＵＵにＬＵ分解されている。そして、各スレッドでＵ１←$LL^{-1}U1$、Ｕ２←$LL^{-1}U2$、Ｕ３←$LL^{-1}U3$、Ｕ４←$LL^{-1}U4$を各スレッドで並列に計算する。この計算の後で、Ｄ１、Ｌ１、Ｌ２、Ｌ３、Ｌ４をローカル域から行列Ａにコピーバックし、バリア同期を取る。更に、Ｃ１←Ｃ１－Ｌ１×Ｕ、Ｃ２←Ｃ２－Ｌ２×Ｕ、Ｃ３←Ｃ３－Ｌ３×Ｕ、Ｃ４←Ｃ４－Ｌ４×Ｕを並列に書くスレッドで行い、最後にバリア同期を取る。

【００３６】図９は、上記処理によって、１段階の処理が終わった後の行列の様子を説明する図である。図９に示されるように、上記処理をすることによって、行列の外側の行及び列が処理されたので、次に、残された左下の部分を同様の方法によって順次処理する。すなわち、ブロック幅iblksを縮小した部分を同じように分割し、図９に示すように、Ｄ１、Ｌ１、Ｌ２、Ｌ３、Ｌ４のブロックに分けて、各スレッドにコピーし、上記と同じ処理を行う。このように、処理を繰り返していくと、最後に２５６×２５６のブロックが残る。この部分は１つのスレッドでＬＵ分解して処理を終了する。

【００３７】なお、上記処理では、ＬＴｉをＬＵ分解するとき、キャッシュメモリ上のデータを効率よく利用するため再帰的なＬＵ分解を利用している。また、Ｃｉ←Ｃｉ－Ｌｉ×Ｕの演算は、キャッシュメモリ上のデータを有効に利用した方法が既知の技術として知られている。

【００３８】図１０は、再帰的ＬＵ分解アルゴリズムを説明する図である。再帰的ＬＵ分解のアルゴリズムはサブルーチンＬＵとして与えられる。ＬＵの取る変数は、図９のアルゴリズムで出てきた、ＬＴｉ（各スレッドでＤ１＋Ｌｉを格納）、ｋ（ＬＴｉの１次元目の大きさ）、iblks （ブロック幅）の他に、ＬＵ分解を始める位置を示すist、ＬＵ分解を行う幅であるnwidである。

【００３９】まず、サブルーチンの最初で、nwidが８、すなわち、ＬＵ分解を行う幅が８であるか否かを判断する。ＹＥＳの場合には、ＬＴｉ（ist ：ｋ、ist ：ist ＋nwid－１）を並列にＬＵ分解する。ここで、ist ：ｋと言う表記は、変数がist からｋまでのＬＴｉを示す意味で、ist ：ist ＋nwid－１というのは、変数がistからist ＋nwid－１までのＬＴｉを示す意味である。以下においても同様である。

【００４０】ＬＴｉのＬＵ分解においては、上記（４）～（１０）の処理を行う。ただし、行の入れ替え部分は、長さiblks で、ＬＴｉ（ｉ、１：iblks ）を入れ替える。また、上記判断がＮＯの場合には、ＬＵ分解を行う幅nwidを２で割った値のＬＵ分解をのＬＵ分解のサブルーチンを再帰的に呼び出して行う。その後、ＴＲＳというルーチンを呼び出す。このルーチンは、ＬＴｉ（ist ：ist ＋nwid／２－１、ist ＋nwid／２：ist ＋nwid）を更新する。更に、ＬＴｉ（ist ：ist ＋nwid／２－１、ist ：ist ＋nwid／２－１）の下三角行列ＬＬを利用して、LL^{-1}をＣｉに左からかけて更新する。次に、ＭＭというルーチンを呼び出す。このルーチンでは、

ＬＴｉ（ist ＋nwid／２：ｋ、ist ＋nwid／２：ist ＋nwid）＝ＬＴｉ（ist ＋nwid／２：ｋ、ist ＋nwid／２：ist ＋nwid）－ＬＴｉ（ist ＋nwid／２：ｋ、ist ：ist ＋nwid／２－１）×ＬＴｉ（ist ：ist ＋nwid／２－１、ist ＋nwid／２：ist ＋nwid）

を演算する。そして、その後、バリア同期を取り、ＬＵ分解のサブルーチンを再帰的に呼び出し、処理した後、処理が終わると、サブルーチンを抜ける。

ｂ）正値対称行列の連立１次方程式の解法

図１１～図１３は、正値対称行列の場合にコレスキー分解を行う処理の概念を説明する図である。

【００４１】実行列の連立１次方程式と同様に、行列をＤ、Ｌ１、Ｌ２、Ｌ３に分割して、各スレッドに対角行列Ｄと更新する列ブロック部分Ｌ１、Ｌ２、Ｌ３を作業

域にコピーして（図１２参照）、独立に並列にして列ブロック部分をコレスキー分解する。なお、この場合、ピボットを取る必要がない。この分解された列ブロックを利用して、小下三角行列（Ｃ１～Ｃ６からなる）を更新する。この更新部分を、並列に行うとき負荷を均等にするために、更新するべき下三角行列をスレッド数を＃Ｔとしたとき、２×＃Ｔに同じブロック幅に分ける（すなわち、Ｃ１とＣ６、Ｃ２とＣ５、Ｃ３とＣ４を組み合わせて処理を行うようにする）。各スレッドはｉ番目及び２×＃Ｔ＋１－ｉ番目のブロックを更新することで負荷を均等にする。

【００４２】現在考えている行列が正値対称行列であるので、図２のＵに対応する部分は、Ｌ１＋Ｌ２＋Ｌ３からなる列ブロックの転置L^Tとなっているので、この場合には、L^T部分は、演算を行う必要が無く、Ｌ１、Ｌ２、Ｌ３をそれぞれ転置してコピーすればよい。

【００４３】図１３は、再帰的コレスキー分解の処理の進行の状況を説明する図である。まず、（１）において、行列の一番左のブロックをコレスキー分解し、３０の部分を３１にコピーする。そして、３１の下の点線で囲まれている部分を斜線部分を用いて更新する。次に、（２）において、処理する列の幅を２倍にして３２の部分を３３にコピーし、３３の下の点線で囲まれた部分を斜線部分を用いて、更新する。そして、（３）に進んで、３４の部分を３５にコピーし、３５の下の点線で示された部分を、斜線部分を用いて更新する。更に、（４）において、３６で示される部分を３７にコピーし、３７の下の部分を斜線部分を用いて更新する。更に、（５）において、３８を３９にコピーし、３９の下の点線で示されている部分を斜線部分を用いて更新し、（６）において、４０を４１にコピーし、斜線部分を用いて、４１の下の部分を更新する。更に、（７）において、４２の部分を４３にコピーし、斜線部分を用いて４３の下の部分を更新する。このように、行列の一部にコレスキー分解を再帰的に繰り返し適用しながら、最終的には行列全体をコレスキー分解する。

【００４４】図１４～図１６は、変形コレスキー分解のアルゴリズムをより詳細に説明する図である。ここでは、説明を簡略化するため４スレッドを使って処理を行う場合を説明する。

【００４５】まず、各スレッドに図１４に示されるＤｘ及びＬｉを連続的な領域ＬＴｉにコピーする。そして、ＬＴｉをＬＤL^T分解する。ＬＤL^T分解は再帰的な方法で行う。そして、ＤＬｉへ以下の計算で値を並列にコピーする。

【００４６】ＤＬｉ←Ｄ×Ｌｉ^T、ここで、ＤはＤｘの対角要素であり、また、右辺は各スレッドのローカル域。そして、Ｄｘ（スレッド１番から）他のＬｉは並列に、もとの領域にコピーバックする。そして、ここで、バリア同期を取り、図１５に示されるように、Ｃ１とＣ８、Ｃ２とＣ７、Ｃ３とＣ６、Ｃ４とＣ５をペアとして、各スレッドで並列に更新する。すなわち、

・スレッド１では、
Ｃ１←Ｃ１－Ｌ１１×ＤＬ１１
Ｃ８←Ｃ８－Ｌ４２×ＤＬ４２
・スレッド２では、
Ｃ２←Ｃ２－Ｌ１２×ＤＬ１２
Ｃ７←Ｃ７－Ｌ４１×ＤＬ４１
・スレッド３では、
Ｃ３←Ｃ３－Ｌ２１×ＤＬ２１
Ｃ６←Ｃ６－Ｌ３２×ＤＬ３２
・スレッド４では、
Ｃ４←Ｃ４－Ｌ２２×ＤＬ２２
Ｃ５←Ｃ５－Ｌ３１×ＤＬ３１
という演算を行う。

【００４７】そして、ここまでの演算が終わった時点でバリア同期を取り、１周り小さくなった行列の領域に関して同じ処理を行う。以上を繰り返し、最後は１つのスレッドで、ＬＤL^T分解して処理を終了する。

【００４８】上記、各スレッドで行うＣｉの更新処理をより一般的な言葉で述べると、スレッド数を＃ＴＨＲＥＡＤとして、Ｌを２×＃ＴＨＲＥＡＤ個に分割し、Ｃの下三角部分も同じく２×＃ＴＨＲＥＡＤ個に分割する。そして、上と下から＃ＴＨＲＥＡＤ個のペアを作り、このペアでＣの分割した部分を更新するという処理になる。

【００４９】図１６は、ＬＤL^T分解の再帰的アルゴリズムを示す図である。ＬＤL^T分解のアルゴリズムは、サブルーチンＬＤＬとして実現される。サブルーチンが取る変数は、前述のＬＵ分解の場合と同様である。

【００５０】まず、nwidが８の場合には、（２０）の部分で、直接ＬＤL^T分解を行う。そして、ＬＴｉ（ist＋８：ｋ、ist ：ist ＋７）を更新する。このとき、ＬＴｉ（ist ：ist ＋７、ist ：ist ＋７）の上三角部分にＤL^Tが入っているので、$(DL^T)^{-1}$を右からかけることによって更新する。

【００５１】（２０）の最初のＩＦ文で、nwidが８でないと判断された場合には、nwid／２を新たなnwidとしてサブルーチンＬＤＬを呼び出し、実行する。ここで、ＬＴｉ（ist ：ist ＋nwid／２－１、ist ＋nwid／２：ist ＋nwid－１）にＤL^Tをコピーする。Ｄは、ＬＴｉ（ist ：ist ＋nwid／２－１、ist ：ist ＋nwid／２－１）の対角要素であり、Ｌは、ＬＴｉ（ist ＋nwid／２：ist ＋nwid－１、ist：ist ＋nwid／２－１）であり、このＬを転置する。

【００５２】そして、ＬＴｉ（ist ＋nwid／２：ｋ、ist ＋nwid／２：ist ＋nwid－１）を更新する。すなわち、
ＬＴｉ（ist ＋nwid／２：ｋ、ist ＋nwid／２：ist ＋nwid－１）＝ＬＴｉ（ist ＋nwid／２：ｋ、ist ＋nwid

／２：ist ＋nwid－１）－ＬＴｉ（ist ＋nwid／２：ｋ、ist ：ist ＋nwid－１）×ＬＴｉ（ist ：ist ＋nwid／２－１、ist ＋nwid／２：ist ＋nwid－１）の演算を行う。

【００５３】次に、ＬＤＬ^{T}分解のサブルーチンＬＤＬを再帰的に呼び出す。そして、処理が終わったら、サブルーチンを抜ける。なお、本発明の実施形態は、上記説明から分かるように、共有メモリ型スカラ並列計算機のアルゴリズムとして与えられるので、このアルゴリズムをプログラムとして実現することになる。あるいは、該並列計算機をＬＵ分解専用機あるいはコレスキー分解専用機として使用する場合には、ＲＯＭなどにプログラムを書き込んでおくことも可能であるが、汎用の並列計算機として使用する場合には、本発明の実施形態のアルゴリズムは、ＣＤ－ＲＯＭ等の可搬記録媒体や、ハードディスクなどの記録媒体にプログラムとして記録しておき、必要に応じて、プログラムをプロセッサにロードして使用する形態が考えられる。

【００５４】このような場合、本発明の実施形態のアルゴリズムを実現するプログラムは、可搬記録媒体などを使って、ユーザに配布が可能である。

（参考文献）

・ＬＵ分解の文献は、１）、２）、変形コレスキー分解の文献は２）

１）P.AMESTOY, M.DAYDE, and I.DUFF, “Use of computational kernelsin the solution of full and sparse linear equations”, M.COSNARD, Y.ROBERT, Q.QUINTON, and M.RAYNAL, PARALLEL&:DISTRIBUTED ALGORITHMS, North-Holland, 1989, pp.13-19

２）G.H.Golub, C.F.van Loan, “Matrix Computations”, second edition, The Johns Hopkins University Press, 1989

・日本語の文献では、以下のものにＬＵ分解及びＬＤＬ^{T}分解の解説がある。

・「数値解析」森　正武著、共立出版会社

・「スーパーコンピュータとプログラミング」島崎眞昭著、共立出版社

（付記１）複数のプロセッサモジュールを持つ共有メモリ型スカラ並列計算機の行列演算において、行列を小行列ブロックに分けるブロック化ステップと、該小行列ブロックの内、対角ブロックと対角ブロックでない小行列ブロックとを該複数のプロセッサモジュールのローカルメモリ領域に格納する格納ステップと、該複数のプロセッサモジュールが並列に、それぞれ有するブロックを演算することにより、複数のプロセッサモジュールで、対角ブロックを冗長に演算する演算ステップと、該演算ステップで得られた小行列ブロックの演算結果を使って、該行列を更新する更新ステップと、を備えることを特徴とする並列行列処理方法を情報装置に実現させるプログラムを格納した、情報装置読み取り可能な記録媒体。

【００５５】（付記２）該行列演算は、行列のＬＵ分解であることを特徴とする付記１に記載の記録媒体。

（付記３）前記複数のプロセッサモジュールが有する小行列ブロックのデータからそれぞれがピボットの候補を抽出する抽出ステップと、該ピボットの候補から該複数のプロセッサモジュールに共通のメモリ領域において、データ値が最大値を示すピボット候補を最終的なピボットと決定するピボット決定ステップと、を更に備え、該決定されたピボットを用いてＬＵ分解を行うことを特徴とする付記２に記載の記録媒体。

【００５６】（付記４）前記ＬＵ分解は、前記行列の外側から再帰的なアルゴリズムによって順次行列の更新を行い、前記行列の内、最後に更新し残った部分を、１つのプロセッサモジュールでＬＵ分解することにより、該行列全体についてＬＵ分解を完了することを特徴とする付記２に記載の記録媒体。

【００５７】（付記５）該行列演算は、行列のコレスキー分解あるいは変形コレスキー分解であることを特徴とする付記１に記載の記録媒体。

（付記６）前記コレスキー分解あるいは変形コレスキー分解は、前記行列の外側から再帰的なアルゴリズムによって順次行列の更新を行い、前記行列の内、最後に更新し残った部分を、１つのプロセッサモジュールでＬＵ分解することにより、該行列全体についてＬＵ分解を完了することを特徴とする付記２に記載の記録媒体。

【００５８】（付記７）前記更新ステップにおいて、更新すべき小行列ブロックの三角行列部分を前記複数のプロセッサモジュールの数の２倍の数のブロックに分割し、該分割された三角行列部分のブロックを２つずつ組み合わせて、各プロセッサモジュールのローカルメモリ域に格納し、演算をプロセッサモジュールに行わせることを特徴とする付記５に記載の記録媒体。

【００５９】（付記８）複数のプロセッサモジュールを持つ共有メモリ型スカラ並列計算機の行列演算において、行列を小行列ブロックに分けるブロック化ステップと、該小行列ブロックの内、対角ブロックと対角ブロックでない小行列ブロックとを該複数のプロセッサモジュールのローカルメモリ領域に格納する格納ステップと、該複数のプロセッサモジュールが並列に、それぞれ有するブロックを演算することにより、複数のプロセッサモジュールで、対角ブロックを冗長に演算する演算ステップと、該演算ステップで得られた小行列ブロックの演算結果を使って、該行列を更新する更新ステップと、を備えることを特徴とする並列行列処理方法。

【００６０】（付記９）複数のプロセッサモジュールを持つ共有メモリ型スカラ並列計算機において、行列を小行列ブロックに分けるブロック化手段と、該小行列ブロックの内、対角ブロックと対角ブロックでない小行列ブロックとを該複数のプロセッサモジュールのローカルメ

モリ領域に格納する格納手段と、該複数のプロセッサモジュールが並列に、それぞれ有するブロックを演算することにより、複数のプロセッサモジュールで、対角ブロックを冗長に演算する演算手段と、該演算ステップで得られた小行列ブロックの演算結果を使って、該行列を更新する更新手段と、を備えることを特徴とする並列行列処理装置。

【００６１】

【発明の効果】本発明によれば、高性能かつスケーラビリティのある行列の処理方法が得られる。

【図面の簡単な説明】

【図１】共有メモリ型スカラ並列計算機のハードウェア構成例を示す図である。

【図２】本発明の実施形態に従ったＬＵ分解の並列処理の概念を説明する図（その１）である。

【図３】本発明の実施形態に従ったＬＵ分解の並列処理の概念を説明する図（その２）である。

【図４】本実施形態のＬＵ分解の処理の流れを示す概略フローチャートである。

【図５】本実施形態のＬＵ分解の方法をより詳細に説明する図（その１）である。

【図６】本実施形態のＬＵ分解の方法をより詳細に説明する図（その２）である。

【図７】本実施形態のＬＵ分解の方法をより詳細に説明する図（その３）である。

【図８】本実施形態のＬＵ分解の方法をより詳細に説明する図（その４）である。

【図９】本実施形態のＬＵ分解の方法をより詳細に説明する図（その５）である。

【図１０】本実施形態のＬＵ分解の方法をより詳細に説明する図（その６）である。

【図１１】正値対称行列の場合にコレスキー分解を行う処理の概念を説明する図（その１）である。

【図１２】正値対称行列の場合にコレスキー分解を行う処理の概念を説明する図（その２）である。

【図１３】正値対称行列の場合にコレスキー分解を行う処理の概念を説明する図（その３）である。

【図１４】変形コレスキー分解のアルゴリズムをより詳細に説明する図（その１）である。

【図１５】変形コレスキー分解のアルゴリズムをより詳細に説明する図（その２）である。

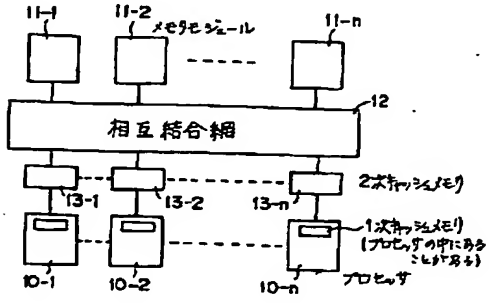
【図１６】変形コレスキー分解のアルゴリズムをより詳細に説明する図（その３）である。

【符号の説明】

１０－１～１０－ｎ　　プロセッサ
１１－１～１１－ｎ　　メモリモジュール
１２　　相互結合網
１３－１～１３－ｎ　　２次キャッシュメモリ

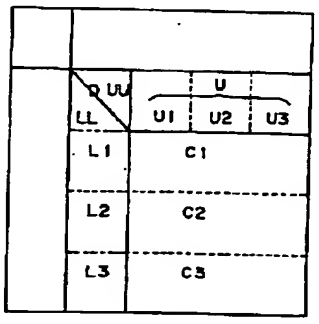
【図１】

共有メモリ型スカラ並列計算機ハードウェア構成例を示す図



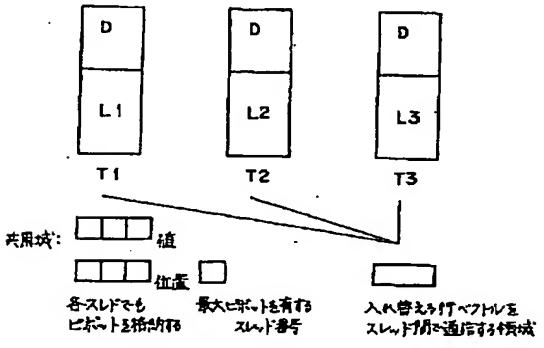
【図２】

本発明の実施形態に従ったLU分解の並列処理の概念を説明する図(その1)



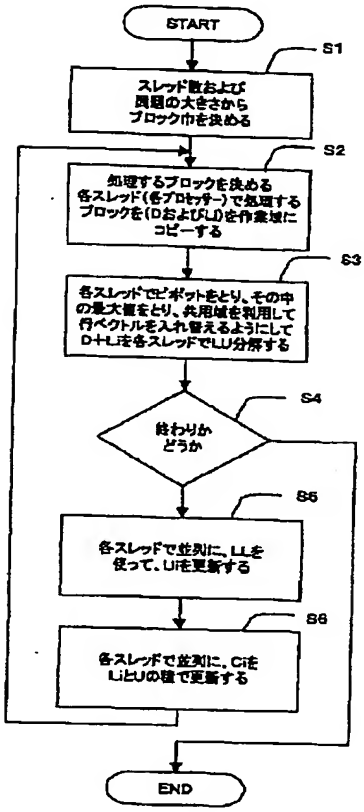
【図３】

本発明の実施形態に従ったLU分解の並列処理の概念を説明する図（その2）



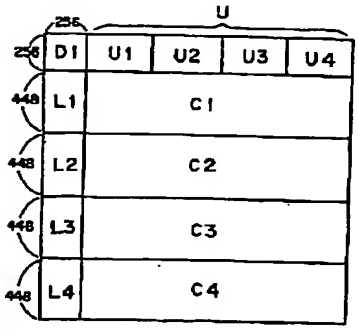
【図４】

本実施形態のLU分解の処理の流れを示す概略フローチャート



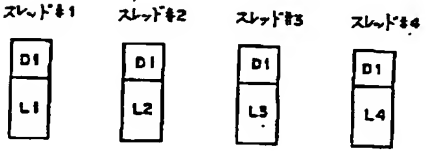
【図５】

本実施形態のLU分解の方法をより詳細に説明する図（その1）



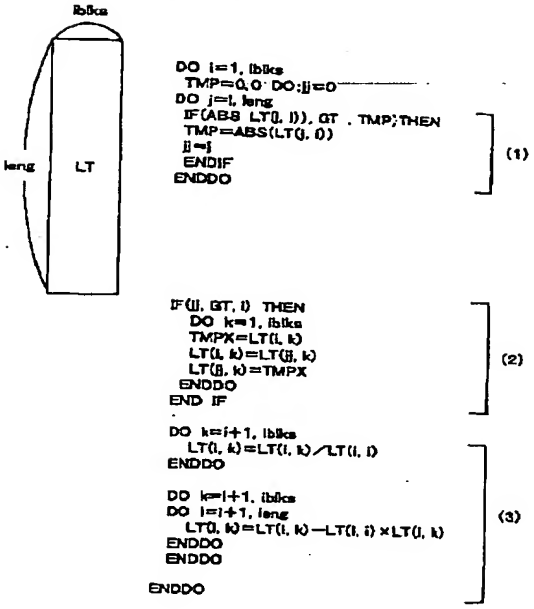
【図６】

本実施形態のLU分解の方法をより詳細に説明する図（その2）



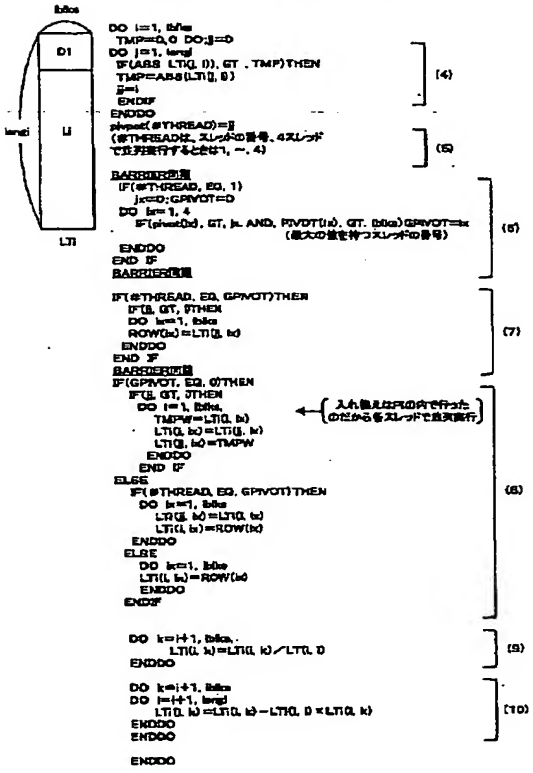
【図７】

本実施形態のLU分解の方法を
より詳細に説明する図(その3)



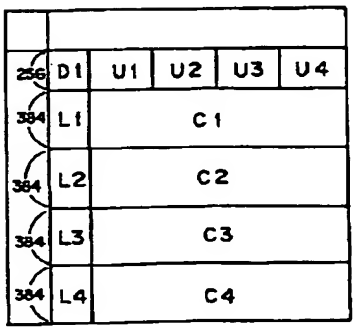
【図８】

本実施形態のLU分解の方法をより詳細に説明する図(その4)



【図９】

本実施形態のLU分解の方法をより詳細に
説明する図(その5)



【図１０】

本実施形態のLU分解の方法を
より詳細に説明する図(その6)

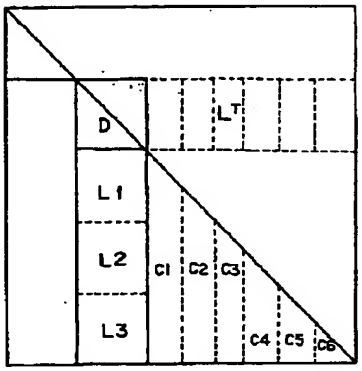
```
subroutine LU(LTI, k, ibks, ist, nwid)
  (LTI)は各スレッドでD1＋Uを格納.
   k:LTIの1次元目の大きさ、
   ibks:ブロック巾
   ist:Lu分解を始める位置、
   nwid:Lu分解を行う巾)
 IF(nwid, eq, 8)Then(巾Bが最小)
LTI(ist:k, ist, ist+nwid−1)を並列にLU分解する。
  ここで、(4)～(10)を行う。
  ただし行の入れ換え部分は長さibksでLTI(1, 1, ibks)を入れ換える
else
  call LU(LTI, k, ibks, ist, nwid/2)
  call TRS( )
  LTI(ist:ist+nwid/2−1, ist+nwid/2:ist+nwid)
  を更新する。LTI(ist:ist+nwid/2−1, ist:ist+nwid/2−1)
の下三角行列Lを利用して、L-1を左から掛けて更新

  call MM( )
  LTI(ist+nwid/2:k, ist+nwid/2:ist+nwid)
  =LTI(ist+nwid/2:k, ist+nwid/2:ist+nwid)
   −LTI(ist+nwid/2:k, ist:ist+nwid/2−1)×
    LTI(ist:ist+nwid/2−1, ist+nwid/2:ist+nwid)

Standard同期
call LU(LTI, k, ibks, ist+nwid/2, nwid/2
end if
  return
  end subroutine
```

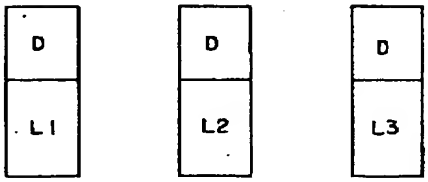
【図１１】

正値対称行列の場合にコレスキー分解を行う処理の概念を説明する図（その1）



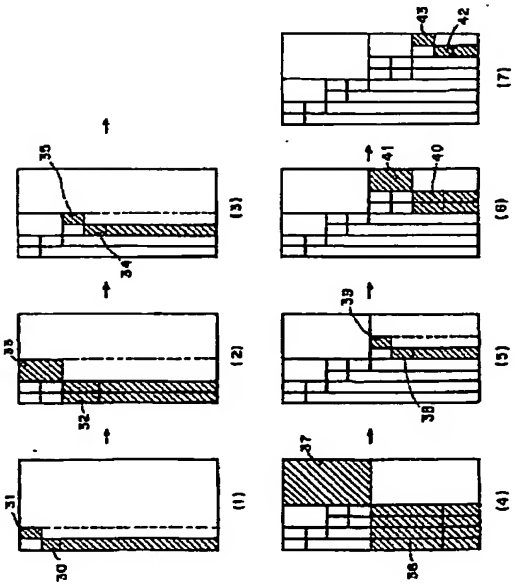
【図１２】

正値対称行列の場合にコレスキー分解を行う処理の概念を説明する図（その2）



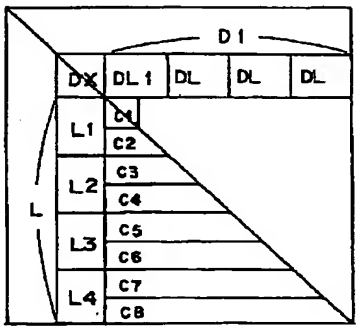
【図１３】

正値対称行列の場合にコレスキー分解を行う処理の概念を説明する図（その3）



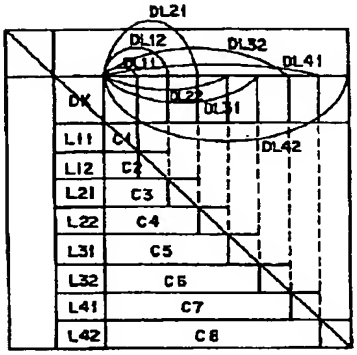
【図１４】

変形コレスキー分解のアルゴリズムをより詳細に説明する図（その1）



【図１５】

変形コレスキー分解のアルゴリズムをより詳細に説明する図(その2)



【図１６】

変形コレスキー分解のアルゴリズムをより詳細に説明する図(その3)

```
subroutine LTD(LTI, k, ibika, ist, nwid)
IF(nwid, EQ, 8)THEN   (ゆ8が最小)
 DOi=ist, ist+7
  DOj=i+1, ist+7
   LTI(i, j)=LTI(j, i)
   LTI(j, i)=LTI(j, i)/LTI(i, i)
  ENDDO
  DO jy=i+1, ist+7
  DO jx=jx, ist+7
   LTI(jx, jy)=LTI(jx, jy)−LTI(jx, i)×LTI(i, jy)
  ENDDO
  ENDDO                                          (20)

 LTI(LTI(ist+8:k, ist:ist+7)を更新
 LTI(LTI(ist:ist+7, ist:ist+7)の上三角にDL^Tが入っているので
   (PL^T)^-1を右からかけて更新する

  ELSE
  call LDL(LTI, k, ibika, ist, nwid/2)

  ・LTI(ist:ist+nwid/2−1, ist+nwid/2:ist+nwid−1)
  にDL^Tをコピーする。(DはLTI(ist:ist+nwid/2−1, ist:ist+nwid/2−1)
  の対象要素、Lは
  LTI(ist+nwid/2:ist+nwid−1, ist:ist+nwid/2−1)
  このL^Tを転置する)

   ・LTI(ist+nwid/2:k, ist+nwid/2:ist+nwid−1)を更新

  LTI(ist+nwid/2:k, ist+nwid/2:ist+nwid−1)
   =LTI(ist:ist+nwid/2:k, ist+nwid/2:ist+nwid−1)−
  LTI(ist+nwid/2:k, ist:ist+nwid−1)×
  LTI(ist:ist+nwid/2−1, ist+nwid/2:ist+nwid−1)

   CALL LDL (LTI, k, ibika, ist+nwid/2, nwid/2)

  ENDIF

   RETURN

   END
```